2025/12/18 16:22 1/4 Teil 2

Teil 2

weiter geht es mit dem **DPB**

das folgende Beispiel stammt aus einer CP/A-Implementation. Bei CP/A erfolgt die Sektorzählung ab 1, deshalb steht in sectran ein inc hl. Die Zählung ab 1 muss bei den direkten Zugriffen beachtet werden!

Beispiel 2

Wir wollen ein RAM-Floppy ansteuern. Die RAM-Floppy (NANOS) hat folgende Eigenschaften:

- 256 K Gesamtkapazität
- die RAM-Floppy kann einen Speicherbereich von 256 Byte in den Hauptspeicher einblenden

Ein Byte mit Adresse A17..A0 in der RAM-Floppy wird so angesprochen:

- 1. Ausgabe A17..A16 auf Port "Bank",
- 2. Ausgabe A15..A8 auf Port "HiAdr",
- 3. Einblenden in den Hauptspeicher (auf Adresse "Window" bis "Window"+255,
- 4. Zugriff auf "Window"+A7..A0

Für die Nutzung im CP/M soll außerdem eine Kopie von CCP+BDOS (5 KByte) auf der RAM-Disk gehalten werden, sinnvollerweise in Systemspuren.

Eine RAM-Floppy hat keine physischen Spuren, deshalb kann man die Aufteilung in virtuelle Spuren und Sektoren nach eigenen Ideen vornehmen.

Die Ansteuerung als Übersicht:

Variante 1

Die Fenstergröße von 256 Byte bietet es an, die Spurgröße als 256 Byte zu wählen. Hi-Byte und Lo-Byte der Tracknummer sind dann direkt "Bank" und "HiAdr". Das macht die Ansteuerung besonders einfach.

also:

```
1 Track = 256 Byte (Fenstergröße)
d.h. 2 Records/track
wir brauchen damit 1600h/256 = 22 Tracks f. Systemspur
insg. 1024 tracks → DSM = 1023-22 = 1011
wir wählen die kleinstmögliche Blockgröße 2k (1k gehen nicht wg. EXM, da DSM > 255)
und z.B. 128 Dir-Einträge (d.h. 2 Dir-Blöcke)
```

Für den DPB ergibt sich damit:

```
;DISKDEF 0,1,2,,2048,1012,128,0,22
                         ;SPT sectors per track
dpb00:
        dw
    db
          4
                     ;BSF block shift factor
                     ;BLM block mask
    db
          15
    db
                     ;EXM null mask
    dw
          1011
                       ;DSM disk size-1
                       ;DRM directory max
    dw
          127
    db
          C<sub>0</sub>h
                      ;ALO alloc 0
                     ;All alloc 1
    db
          0
                     ;CKS check size
    dw
          0
    dw
          22
                      ;OFS track offset
alv00:
        ds
               007Fh
                            ;allocation vector
csv00:
               0000
                                 ;check vector
        ds
```

Die BIOS-Routinen zum Blocklesen und -schreiben verweisen auf folgende Routinen. Wegen der Spurgröße von 256 Byte = 2 Records muss ein Blocking/Deblocking erfolgen. Glücklicherweise ist das bei einer RAM-Disk nicht weiter schwierig umzusetzen, da innerhalb des Zugriffsfensters nur der angesprochene Bereich von 128 Byte gelesen bzw. verändert wird.

```
; Lesen von Diskette
READ: CALL ADRE
READ1: LDIR
OUT (READDI), A
```

2025/12/18 16:22 3/4 Teil 2

```
0UT
            (RAMDI), A
    X0R
    RET
; Schreiben auf Diskette
WRITE:
        CALL
                 ADRE
          DE, HL
    ΕX
    JR
          READ1-#
; Berechnung Adr.
ADRE:
        0UT
                (RAMEN), A
    0UT
            (READEN), A
    LD
          HL, (TRACK)
    0UT
            (LDAH), L
                          ; hi-adr.
    0UT
            (LDBB), H
                          ; Bank
    LD
                          ; das ist eine xx00h-Adr.
          HL, WINDOW
    LD
                          ; 1 oder 2 (in CP/A wg. SECTRAN)
          a, (SECTOR)
    CP
          2
    jr
          nz, ADRE0a
          L,80h
    LD
ADREa:
               DE, (DMAAD)
        LD
    LD
          BC,128
    RET
```

Variante 2

Um eine kleinere Blockgröße nutzen zu können, muss die Anzahl der Spuren ≤ 256 werden. Da geht z.B. mit einer Spurgröße von 2 KByte.

```
1 Track = 2048 Byte
d.h. 16 Records/Track
wir brauchen damit 1600h/2048 = 3 Tracks f. Systemspur
insg. 256 Tracks→ DSM = 255-3
kleinste Blockgröße 1k
und z.B. 64 Dir-Einträge (d.h. 2 Dir-Blöcke)
```

Diese Aufteilung ist aufgrund der kleineren Blockgröße günstiger, wenn viele kleine Dateien auf der RAM-Disk gehalten werden sollen. Auch wird weniger Platz für den Allocation Vektor ALVxx benötigt. Aber die Umrechnung logischer Track-Sektor → Adr. f. RAM-Disk ist aufwendiger!

```
;DISKDEF 1,1,16,,1024,252,64,0,3
dpb01:
                           ;SPT sectors per track
         dw
           3
                     ;BSF block shift factor
    db
    db
           7
                     ;BLM block mask
    db
           0
                     ;EXM null mask
           251
                       ;DSM disk size-1
    dw
                      ;DRM directory max
    dw
           63
           C<sub>0</sub>H
                       ;ALO alloc O
    db
    db
           0
                     ;All alloc 1
```

```
dw 0 ;CKS check size
dw 3 ;OFS track offset
;
alv01: ds 0020h ;allocation vector
csv01: ds 0000h ;check vector
```

Read und Write sind wie oben implementiert, die Adressierung ist jetzt umfangreicher:

```
ADRE:
        0UT
               (RAMEN), A
   0UT
           (READEN), A
    ;Adr. Fenster = (track*16+sector)/2
   LD
          HL, (TRACK)
          HL,HL
   ADD
   ADD
           HL, HL
   ADD
          HL,HL
   ADD
          HL,HL
                        ; HL = Track * 10h (SPT)
   LD
          DE, (SECTOR)
   DEC
           DE
                     ; wg. CP/A
   ADD
           HL, DE
                       ; HL := HL + Sector
   X0R
          Α
                    ; A = 0, Cy = 0
   RR
          Н
   RR
          L
                   ; HL := HL/2 ( da 2 Sektoren/Fenster )
   RR
                   ; L Bit0 nach A Bit7 ( A = 0 oder 80h)
   0UT
           (LDAH), L ; hi-adr.
   0UT
           (LDBB), H
                        ; Bank
   LD
          H, Hi(WINDOW)
   LD
          L,A
   LD
          DE, (DMA)
          BC,128
   LD
   RET
```

From:

https://hc-ddr.hucki.net/wiki/ - Homecomputer DDR

Permanent link:

https://hc-ddr.hucki.net/wiki/doku.php/cpm/write_a_bios/teil_2?rev=1517468245

Last update: 2018/02/01 06:57

